Programming Fundamentals using Java
Abstract Classes
Abstract classes

From the previous lecture:

```java
public class GeometricObject {
    protected String Color;
    protected String name;
    protected float area;

    // Constructors...

    // get/set methods...
}
```
Abstract classes

- So we can do:

```java
GeometricObject gObj = new GeometricObject("AnObject, Red");
```
Abstract classes

- So we can do:

```java
GeometricObject gObj = new GeometricObject("AnObject, Red");
```

But does it make sense to do this?

What is a “GeometricObject” by itself?
Abstract classes

Solution:

Make it **abstract**!
Abstract classes

- Solution:

- Make it **abstract**!

```
public abstract class GeometricObject {
```
Abstract classes

- Used for defining classes for “abstract” concepts. (‘GeometricObject’, ‘Animal’, etc.)
Abstract classes

- Used for defining classes for “abstract” concepts. (‘GeometricObject’, ‘Animal’, etc.)

- Then define “concrete” concepts as subclasses. (Circle, Rectangle, etc.)
Abstract classes

- Used for defining classes for “abstract” concepts. (‘GeometricObject’, ‘Animal’, etc.)

- Then define “concrete” concepts as subclasses. (Circle, Rectangle, etc.)

- More strictness = less room for ambiguity/error.
Abstract classes

- Every **GeometricObject** has an area.
Abstract classes

- Every *GeometricObject* has an area.

- But *getArea()* defined differently for *concrete* objects.
Abstract classes

- Every **GeometricObject** has an area.

- But **getArea()** defined differently for **concrete** objects.

- (Not defined for the **GeometricObject** class)
Abstract classes

// Circle
def class Circle extends GeometricObject {
def public float getArea() {
def return Math.PI * radius * radius;
def}
def}
Abstract classes

// Circle
public class Circle extends GeometricObject {
    public float getArea() {
        return Math.PI * radius * radius;
    }
}

// Rectangle
public class Rectangle extends GeometricObject {
    public float getArea() {
        return width * height;
    }
}
Abstract classes

```java
GeometricObject[] objs = new GeometricObject[2];
objs[0] = new Circle(3);
objs[1] = new Rectangle(1, 2);

for (GeometricObject i : objs) {
    System.out.println(i.getArea());
}
```
Abstract classes

```java
GeometricObject[] objs = new GeometricObject[2];
objs[0] = new Circle(3);
objs[1] = new Rectangle(1, 2);

for (GeometricObject i : objs) {
    System.out.println(i.getArea());
}
```

Will not Compile! Cannot find “getArea()” in the GeometricObject class.
Abstract classes

- So make `getArea()` an *abstract method* in `GeometricObject`. 
Abstract classes

- So make `getArea()` an *abstract method* in `GeometricObject`.

- Only a declaration in `GeometricObject`:

```java
public abstract class GeometricObject {
    ...

    public abstract float getArea();
}
```
Abstract classes

```java
GeometricObject[] objs = new GeometricObject[2];
objs[0] = new Circle(3);
objs[1] = new Rectangle(1, 2);

for (GeometricObject i : objs) {
    System.out.println(i.getArea());
}

Will now work! getArea() defined in GeometricObject
```
Abstract class characteristics

- Class has *abstract* methods = has to be an *abstract* class
Abstract class characteristics

- Class has *abstract* methods = has to be an *abstract* class

- **But:** *Abstract* class can have **no** *abstract* methods.
Abstract class characteristics

- Class has abstract methods = has to be an abstract class

- But: Abstract class can have no abstract methods.

- Subclass can be abstract, superclass can be concrete.
Abstract class characteristics

- **Concrete** subclasses **must** define superclass **abstract** methods.
  - Circle, Rectangle have to define their own `getArea();`

- **Abstract** methods can be defined, but usually aren’t.

- **Abstract** classes cannot be instantiated.
Abstract class characteristics

- *Abstract* class can be used as a data type though.

- **Example:**

  ```java
  GeometricObject gObj;
  
  GeometricObject[] objs = new GeometricObject[2];
  ```
Abstract class characteristics

Why is this allowed:

```
GeometricObject[] objs = new GeometricObject[2];
```

They can’t be instantiated, so why allow as data types?
GeometricObject[] objs = new GeometricObject[2];
objs[0] = new Circle(3);
objs[1] = new Rectangle(1, 2);

for (GeometricObject i : objs) {
    System.out.println(i.getArea());
}

Can take advantage of polymorphism!
Abstract classes

- Let’s say we need to check if areas of GeometricObjects are equal
Abstract classes

- Let’s say we need to check if areas of GeometricObjects are equal

- Define method called equalArea(...)
public static void main(String[] args) {
    Circle c1 = new Circle(3.0);
    Rectangle r1 = new Rectangle(2, 3);
    Rectangle r2 = new Rectangle(1, 6);

    SOP("c1 and r1 have equal area: " + equalArea(c, r1));
    SOP("r2 and r1 have equal area: " + equalArea(r2, r1));
}
Abstract classes

- How do you define `equalArea(...)`?
Abstract classes

- How do you define `equalArea(...)`?

```java
public static boolean equalArea( ? obj1, ? obj2) {
    return obj1.getArea() == obj2.getArea();
}
```
Abstract classes

- Could define it for all different comparisons:

```java
public static boolean equalArea(Circle c1, Rectangle r1)
{
    return c1.getArea() == r2.getArea();
}
```
Abstract classes

- Could define it for all different comparisons:

```
public static boolean equalArea( Circle c1, Circle c2) {
    return c1.getArea() == c2.getArea();
}
```
Abstract classes

- Could define it for all different comparisons:

```java
public static boolean equalArea(Rectangle r1, Rectangle r2)
{
    return r1.getArea() == r2.getArea();
}
```
Abstract classes

- Could define it for all different comparisons:

```
public static boolean equalArea(Rectangle r1, Rectangle r2)
{
    return r1.getArea() == r2.getArea();
}
```

Tedious...
Abstract classes

- Make it general thanks to *abstract* methods/classes:

```java
public static boolean equalArea( GeometricObject g1, GeometricObject g2 )
{
    return g1.getArea() == g2.getArea();
}
```
Abstract classes

Why not use the **Object** class?

```java
public static boolean equalArea( Object g1, Object g2)
{
    return g1.getArea() == g2.getArea();
}
```

What’s the problem?
Abstract classes

- Why not use the **Object** class?

```java
public static boolean equalArea(Object g1, Object g2) {
    return g1.getArea() == g2.getArea();
}
```

**What’s the problem?**

Object class has no getArea() method!
Abstract classes

Why not use the Object class?

```java
public static boolean equalArea(Object g1, Object g2) {
    return g1.getArea() == g2.getArea();
}
```

What’s the problem?

Object class has no getArea() method!

How would you fix it?
Abstract classes

- Why not use the **Object** class?

```java
public static boolean equalArea(Object g1, Object g2) {
    GeometricObject go1 = (GeometricObject)g1;
    GeometricObject go2 = (GeometricObject)g2;
    return go1.getArea() == go2.getArea();
}
```

Cast the objects
Next Lecture...

- Interfaces.